DS1 : Corrigé

Problème 1

1. On note $G = ((A \Rightarrow B) \wedge (\neg A \Rightarrow B)) \Rightarrow B$. La table de vérité de G est la suivante :

A	B	$A \Rightarrow B$	$\neg A \Rightarrow B$	$(A \Rightarrow B) \wedge (\neg A \Rightarrow B)$	G
V	V	V	V	V	V
V	F	F	V	F	V
V	I	I	V	I	V
F	V	V	V	V	V
F	F	V	F	F	V
F	I	V	I	I	V
I	V	V	V	V	V
I	F	I	I	I	I
I	I	V	V	V	I

   On en déduit que G est satisfiable mais n'est pas une tautologie (alors que c'en est une dans S_2).

2. Dès que v est une trivaluation telle que $v(A) = \text{I}$, on a $v(A \vee \neg A) = \text{I} \neq \text{V}$. Donc cette formule n'est pas une tautologie selon la sémantique S_L alors que c'en est une selon la sémantique S_2. D'où on constate (avec une stupeur contenue) que modifier la sémantique change le sens des formules. Qui l'eut crû ?

3. L'énoncé ne demande pas une tautologie non triviale, on peut donc légitimement proposer comme formule tautologique : $\top$. On peut aussi proposer, par exemple, la formule $x \Rightarrow x$ où x est une variable si on souhaite éviter les tautologies triviales.

   Remarque : Proposez une formule la plus simple possible de sorte à agréer le correcteur.

4. D'après les lois de De Morgan, en sémantique standard on a $G_1 = \neg(\neg A \wedge \neg B) \equiv A \vee B$. Pour savoir si cette équivalence est conservée dans la sémantique S_L, on dresse les tables de vérité de $A \vee B$ et G_1 :

A	B	$\neg A \wedge \neg B$	G_1	$A \vee B$
V	V	F	V	V
V	F	F	V	V
V	I	F	V	V
F	V	F	V	V
F	F	V	F	F
F	I	I	I	I
I	V	F	V	V
I	F	I	I	I
I	I	I	I	I

   Les deux dernières colonnes de cette table étant identiques, l'équivalence $G_1 \equiv A \vee B$ est conservée. Remarquons qu'il n'y a en fait pas besoin de construire les lignes ne faisant intervenir que des valeurs de vérité de la sémantique S_2 puisque l'équivalence de G_1 et $A \vee B$ y est déjà avérée.

5. La formule $G_2 = \neg A \vee B$ convient. On constate que pour la trivaluation v assignant I à A et à B on obtient d'une part $v(G_2) = \text{I}$ et d'autre part $v(A \Rightarrow B) = \text{V}$. En conséquence de quoi ces deux formules ne selon pas équivalentes dans la sémantique S_L.

6. Reformulons la question : il s'agit de savoir si l'équivalence $\neg B \Rightarrow \neg A \equiv A \Rightarrow B$, valide en sémantique standard, le reste selon la sémantique S_L. Pour ce faire, il suffit de comparer les tables de vérité de ces deux formules, et ce uniquement pour les trivaluations faisant intervenir I comme remarqué en question 4 :

A	B	$\neg B \Rightarrow \neg A$	$A \Rightarrow B$
V	I	I	I
F	I	V	V
I	V	V	V
I	I	V	V
I	F	I	I

Ainsi, la contraposition reste sémantiquement vraie dans S_L.

7. a) Redémontrons cette propriété de cours par double implication. Soit $G, H \in \mathcal{F}$.

   - Supposons que $G \models H$ (ce qui est au passage, un abus pour $\{G\} \models H$) et soit v une valuation quelconque. Si $v(G) = \text{V}$, alors $v(H) = \text{V}$ par définition de $G \models H$ et donc $v(G \Rightarrow H) = \text{V}$ d'après la sémantique standard de $\Rightarrow$. Si $v(G) = \text{F}$ alors peut importe la valeur de $v(H)$ on a $v(G \Rightarrow H) = \text{V}$, toujours d'après la sémantique standard de $\Rightarrow$. Donc $\models G \Rightarrow H$ est avéré.
   - Réciproquement, si $\models G \Rightarrow H$, cela signifie que toute valuation satisfait $G \Rightarrow H$. Si v est une valuation qui satisfait G, comme elle satisfait $G \Rightarrow H$, elle satisfait également H d'après la sémantique standard de $\Rightarrow$. Donc $G \models H$ par définition.

   Remarque : On peut directement procéder par équivalences en utilisant le fait que $G \models H$ si et seulement si $G \wedge \neg H$ est non satisfiable.

   b) Considérons deux variables x et y et construisons les formules $G = x \wedge y$ et $H = y$.

   - D'une part, on a $G \models H$. En effet, si v satisfait G, on a nécessairement $v(y) = \text{V}$ d'après la sémantique de $\wedge$ selon S_I et donc v satisfait H.
   - D'autre part, pour la trivaluation v telle que $v(x) = \text{I}$ et $v(y) = \text{F}$, on a $v(G \Rightarrow H) = \text{I} \neq \text{V}$. Donc $G \Rightarrow H$ n'est pas une tautologie dans S_I ce qui contredit le fait que $\models G \Rightarrow H$.

   Par conséquent, l'affirmation $(\star)$ est fausse dans la sémantique S_I.

   c) On modifie la sémantique S_I comme suit : on conserve la sématique de $\neg, \vee$ et $\wedge$ comme demandé par l'énoncé et on modifie celle de $\Rightarrow$ de telle sorte à ce que la sémantique de $x \Rightarrow y$ soit V (à la place de I) lorsque x vaut I et y vaut F. On obtient ainsi une sémantique qu'on nomme S_M et on montre que $(\star)$ est vérifiée dans cette sémantique par double implication :

   - Si $G \models H$, montrons que $G \Rightarrow H$ est une tautologie. Soit v une trivaluation. Si $v(G) = \text{V}$ alors $v(H) = \text{V}$ par hypothèse donc $v(G \Rightarrow H) = \text{V}$ d'après la sémantique de $\Rightarrow$ dans S_M (qui est la même que dans S_2). Si $v(G) \neq \text{V}$ (donc vaut soit I, soit F), on a justement modifié la sémantique de $\Rightarrow$ de telle sorte à ce que quelque soit la valeur de $v(H)$ on ait $v(G \Rightarrow H) = \text{V}$. Donc pour toute trivaluation v, $v(G \Rightarrow H) = \text{V}$ ce qui conclut.
   - Réciproquement, si $\models G \Rightarrow H$, montrons que $G \models H$. Soit donc v une trivaluation satisfaisant G. Par hypothèse, v satisfait $G \Rightarrow H$ puisque c'est une tautologie. La sémantique de $\Rightarrow$ dans S_M nous assure alors qu'on a nécessairement $v(H) = \text{V}$ ce qui conclut.

8. Non. Si $A = x$ où $x \in \mathcal{V}$, la trivaluation v telle que $v(x) = \text{I}$ vérifie $v(A \Rightarrow A) = \text{I} \neq \text{V}$ d'après la dernière ligne de la table de vérité de $\Rightarrow$ selon S_K.

   Remarque : Il arrive néanmoins que $A \Rightarrow A$ soit une tautologie dans S_K ; par exemple lorsque $A = \top$!

9. Montrons pour toute formule $G \in \mathcal{F}_v$, la propriété $P(G)$ suivante : la trivaluation v_I qui assigne I à toutes les variables propositionnelles utilisées dans G vérifie $v_I(G) = \text{I}$. On procède par induction structurelle.

   Cette propriété est immédiatement satisfaite pour les éléments de base de $\mathcal{F}_v$ que sont les variables propositionnelles. Soit à présent deux formules $G, H \in \mathcal{F}_v$ vérifiant respectivement $P(G)$ et $P(H)$ et no-

tons v_I la valuation qui assigne I à toutes les variables intervenant dans G ou dans H. Par hypothèse, $v_I(G) = v_I(H) = \mathrm{I}$ et on en déduit que $v_I(\neg G) = v_I(G \vee H) = v_I(G \wedge H) = v_I(G \Rightarrow H) = \mathrm{I}$ en observant la dernière ligne des tables de vérité des connecteurs $\neg, \vee, \wedge$ et $\Rightarrow$ selon la sémantique S_K.

Il s'ensuit que pour toute formule $G \in \mathcal{F}_v$, il existe une trivaluation qui rend G indéterminée et non vraie, ce qui défend à toute formule de $\mathcal{F}_v$ d'être une tautologie.

10. Tout d'abord, si on a déterminé entièrement la table de vérité de $\wedge$ et celle de $\neg$, le point (3) assure que la table de vérité de $\vee$ est elle aussi déterminée. Mais, une fois qu'on a déterminé la table de vérité de $\vee$ et de $\neg$, la table de vérité de $\Rightarrow$ se déduit d'après le point (4).

    On en déduit que répondre à la question revient à identifier quelles sont les tables de vérité possibles pour $\neg$ et $\wedge$. Or, il n'y a qu'une seule sémantique possible pour $\neg$ car le point (5) impose que $\neg I = I$. Il ne reste donc plus qu'à identifier le nombre de tables de vérité possibles pour $\wedge$.

    Puisque $\wedge$ est commutatif d'après (2) et prolonge S_2 d'après (1), il suffit de déterminer $\mathrm{I} \wedge \mathrm{F}$, $\mathrm{I} \wedge \mathrm{I}$ et $\mathrm{I} \wedge \mathrm{V}$ pour déterminer entièrement la sémantique de $\wedge$. Or, le point (6) garantit la croissance de la fonction $x \mapsto x \wedge \mathrm{F}$, donc $\mathrm{I} \wedge \mathrm{F}$ est nécessairement compris entre $\mathrm{V} \wedge \mathrm{F} = \mathrm{F}$ et $\mathrm{F} \wedge \mathrm{F} = \mathrm{F}$. Donc la seule possibilité est d'avoir $\mathrm{I} \wedge \mathrm{F} = \mathrm{F}$. Pour savoir quelles sont les possibilités pour $\mathrm{I} \wedge \mathrm{I}$ et $\mathrm{I} \wedge \mathrm{V}$, on fait une disjonction de cas selon la valeur de $\mathrm{I} \wedge \mathrm{V}$:

    - Si $\mathrm{I} \wedge \mathrm{V} = \mathrm{F}$, alors la croissance de $x \mapsto x \wedge \mathrm{I}$ et la commutativité de $\wedge$ obligent à avoir $\mathrm{I} \wedge \mathrm{I}$ compris entre $\mathrm{I} \wedge \mathrm{V} = \mathrm{F}$ et $\mathrm{I} \wedge \mathrm{F} = \mathrm{F}$ donc $\mathrm{I} \wedge \mathrm{I} = \mathrm{F}$. On obtient une première sémantique possible.
    - Si $\mathrm{I} \wedge \mathrm{V} = \mathrm{I}$, on raisonne similairement : par croissance, on peut avoir $\mathrm{I} \wedge \mathrm{I} = \mathrm{I}$ ou $\mathrm{I} \wedge \mathrm{I} = \mathrm{F}$. On obtient deux nouvelles sémantiques possibles.
    - Si $\mathrm{I} \wedge \mathrm{V} = \mathrm{V}$, on peut cette fois avoir $\mathrm{I} \wedge \mathrm{I}$ égal à V, I ou F d'où trois nouvelles sémantiques.

    Ainsi, si S est une sémantique vérifiant les points (1) à (6), S est nécessairement l'une des 6 sémantiques décrites ci-dessus. Réciproquement, chacune de ces 6 sémantiques vérifie bien les points (1) à (6) et on en déduit qu'il existe 6 sémantiques prolongeant S_2 selon les contraintes de l'énoncé.

Remarques : Les sémantiques trivaluées peuvent de prime abord ressembler à une bizarrerie de logicien. C'est en fait dans l'idée de modéliser des situations tout à fait concrètes qu'elles ont été introduites. En effet, la sémantique booléenne classique est inadaptée pour parler du futur. "Il y aura un séisme demain" n'est - au moment où cette proposition est énoncée - ni vrai ni fausse mais indéterminée.

On retrouve l'utilisation de logiques trivaluées dans des domaines très concrets comme dans SQL, dans lequel elles servent à manipuler NULL : cette valeur étant interprétée comme étant la valeur de vérité I de l'énoncé.

Problème 2

1. H_1 est satisfiable par la valuation v telle que $v(v_0) = v(v_1) = v(v_2) = 0$.

    La formule H_2 est non satisfiable car il est impossible d'en satisfaire les clauses v_1 et $\neg v_1$ simultanément.

    Supposons par l'absurde qu'il existe une valuation v satisfaisant H_3. On a nécessairement $v(v_1) = 1$ (pour satisfaire la première clause) puis $v(v_0) = 0$ (pour satisfaire la deuxième clause). Mais alors $v(v_0 \vee \neg v_1) = 0$ et on obtient une contradiction avec le fait que v satisfait H_3. Ainsi, H_3 n'est pas satisfiable.

2. Si H contient un littéral négatif dans chacune de ses clauses, la valuation attribuant la valeur faux à toutes les variables intervenant dans H satisfait toutes les clauses de H donc H elle même.

    Remarque : Ce raisonnement est valide quand bien même H contient 0 clauses : en ce cas, H est réduite au neutre pour le connecteur $\wedge$ c'est-à-dire $\top$ qui est satisfiable pour n'importe quelle valuation.

3. Par hypothèse, $H = v \wedge C_2 \wedge ... \wedge C_n$ où les C_i sont des clauses de Horn dont aucune n'est réduite à $\neg v$. Pour tout $i \in [\![2, n]\!]$, on définit une clause C'_i à partir de C_i comme suit :

   - Si v est un littéral de C_i alors $C'_i = \top$.
   - Sinon, C'_i est la clause obtenue à partir de C_i en en supprimant l'éventuel littéral $\neg v$.

   Montrons que la formule $H' = \bigwedge_{\substack{i=2 \\ C'_i \neq \top}}^{n} C'_i$ satisfait toutes les propriétés désirées :

   - H' est une formule de Horn car c'était le cas de H.
   - La variable v n'intervient plus dans H', que ce soit positivement ou négativement.
   - Les formules H et H' sont équisatisfiables. On procède par double implication :

     Si H est satisfiable, il existe une valuation φ telle que $\varphi(H) = 1$. On a nécessairement $\varphi(v) = 1$ puisque φ satisfait toutes les clauses de H. Pour les clauses C'_i non réduites à $\top$, on a par construction $C_i = C'_i \vee \neg v$. Puisque $\varphi(\neg v) = 0$, on a $\varphi(C'_i) = \varphi(C'_i \vee \neg v) = \varphi(C_i) = 1$. Ainsi, v satisfait toutes les clauses de H' donc H' elle même.

     Réciproquement, s'il existe une valuation φ' telle que $\varphi'(H') = 1$, étendons la en une valuation φ telle que $\varphi(v) = 1$ (ce qui est possible puisque v n'intervient pas dans H'). Si $i \in [\![2, n]\!]$, soit C_i contient le littéral v et auquel cas $\varphi(C_i) = 1$, soit C_i ne contient pas le littéral v et dans ce cas C_i est égal à C'_i ou à $C'_i \vee \neg v$ et comme $\varphi(C'_i) = 1$, on a aussi $\varphi(C_i) = 1$. Donc H est satisfiable.

   Remarque : Il y a bien d'autres façons d'exprimer cette construction. Par exemple, H' correspond aussi à la formule $H[\top/v]$ simplifiée par les transformations utilisées dans l'algorithme de Quine.

4. On propose le pseudo-code suivant :

```
Entrée : Une formule de Horn H
Sortie : Oui si H est satisfiable, non sinon
1. hornsat(H) =
2.    Si H = ⊤, renvoyer oui
3.    Sinon, H est de la forme C_1 ∧ ... ∧ C_n
4.       Si tous les C_i contiennent un littéral négatif, renvoyer oui
5.       Sinon, choisir une clause de la forme v où v est une variable
6.          Si une des clauses de H est égale à ¬v, renvoyer non
7.          Sinon, renvoyer hornsat(H') avec H' définie en question 3
```

   La terminaison de cet algorithme est garantie par le fait que le nombre de variables dans la formule H décroît strictement dans l'ensemble bien fondé $\mathbb{N}$ au fil des appels récursifs, d'après la question 3. On atteint donc toujours le cas de base couvert par la ligne 2.

   La correction en ligne 4 est assurée par la question 2. La correction en ligne 6 est assurée par le fait que $v \wedge \neg v$ est antilogique. La correction dans le cas récursif est donnée par la question 3. On remarque que le si... sinon des lignes 4 et 5 couvre bien tous les cas grâce au fait que H est une formule de Horn : en effet, si une clause de H ne contient pas de littéral négatif, elle est nécessairement réduite à une variable.

5. L'analyse peut dépendre de la façon dont on considère qu'une formule de Horn est représentée. Ici, on choisit de considérer un littéral comme un couple (i, b) où i est un entier symbolisant le numéro de la variable du littéral et b est un booléen (vrai si le littéral est positif, faux si le littéral est négatif). Une clause est représentée par une liste de littéraux et une formule de Horn par une liste de clauses.

Notons $C(n,m)$ le nombre d'opérations au pire cas effectuées sur une formule H à m clauses et comptant n variables distinctes. Remarquons que ce pire cas correspond au cas où la formule contient un littéral réduit à une variable v mais n'a pas de clause égale à $\neg v$.

Tester si une clause est réduite à une variable se fait en $O(1)$: il suffit de vérifier que la clause est une liste contenant un seul couple et que le deuxième argument de ce couple est vrai. Ainsi, tester si toutes les clauses contiennent un littéral négatif et trouver une clause ne contenant qu'une variable le cas échéant peut se faire d'un coup d'un seul en $O(m)$ (en effet, si une clause contient plus de deux éléments, d'après la définition d'une clause de Horn, l'un des deux au moins est un littéral négatif).

Dans le pire cas, il y a effectivement une clause réduite à une variable v et $O(m)$ opérations sont alors nécessaires pour vérifier que la clause $\neg v$ n'est pas présente. Dans le pire cas, $\neg v$ n'est effectivement pas présente et on doit alors construire la formule H' ce qui peut se faire en $O(nm)$ (on parcourt m clauses qui sont des listes de taille au plus $2n$ puisque les littéraux d'une clause sont distincts). On applique ensuite l'algorithme à une formule comptant $m-1$ clauses et $n-1$ variables. Ainsi :

$$C(n,m) \leq Knm + C(n-1,m-1)$$

où K est une constante positive. Par croissance de la complexité en fonction de m, on a $C(n-1,m-1) \leq C(n-1,m)$, ce qui permet de réécrire l'inéquation ci-dessus de manière plus facilement exploitable :

$$C(n,m) \leq Knm + C(n-1,m)$$

Ainsi,

$$\begin{aligned} C(n,m) &\leq Knm + K(n-1)m + K(n-2)m + ... + C(0,m) \text{ en exploitant l'inégalité ci-dessus} \\ &\leq \sum_{i=1}^{n} Kmi + O(m) \text{ car tester si les } m \text{ éléments d'une liste sont des listes vides se fait} \\ &\qquad\qquad\qquad\qquad \text{linéairement en } m \\ &= O(mn^2) \end{aligned}$$

Il est possible selon vos choix que vous ne trouviez pas exactement cette complexité mais dans tous les cas elle doit être un polynôme en n et m.

6. La question 5 prouve que la satisfiabilité d'une formule de Horn peut être décidée en temps polynomial en sa taille : le problème HORN-SAT est dans P, contrairement à SAT qui lui est NP-complet. Autrement dit : HORN-SAT est un problème bien plus facile que SAT.

7. C'est immédiat :

$$\begin{aligned} (v_0 \wedge ... \wedge v_k) \Rightarrow v &\equiv \neg(v_0 \wedge ... \wedge v_k) \vee v \\ &\equiv \neg v_0 \vee ... \vee \neg v_k \vee v \text{ d'après les lois de De Morgan et l'associativité de } \vee \end{aligned}$$

8. Si la clause considérée ne contient pas de littéral positif, il s'agit juste de vérifier si l'une de variables intervenant dans les littéraux négatifs est assignée à faux. Si un littéral positif est présent, on vérifie en plus s'il est assigné à vrai. D'où :

```
let evalue (clause:hornclause) (v:valuation) = match clause.csqc with
  |None -> List.exists (fun i -> not v.(i)) clause.hyp
  |Some c -> v.(c) || (List.exists (fun i -> not v.(i)) clause.hyp)
```

9. On procède récursivement en utilisant la fonction evalue : si la première clause de la liste est rendue fausse par la valuation, on la renvoie ; sinon, on cherche une clause fausse dans le reste de la liste :

```
let rec trouve_clause_fausse (l:hornclause list) (v:valuation) :hornclause option =
    match l with
    |[] -> None
    |c::cl -> if not (evalue c v) then Some c
              else trouve_clause_fausse cl v
```

10. On commence par récupérer le nombre de variables intervenant dans la formule F de sorte à initialiser une valuation de la bonne taille. On introduit une variable pas_sat (respectivement, sat) dont la sémantique est la suivante : pas_sat (respectivement, sat) vaut true si on est certain que F n'est pas satisfiable (respectivement, est satisfiable). La sémantique de la boucle tant que est donc la suivante : tant qu'on ne sait pas trancher si F est satisfiable ou non, on adapte la valuation v pour essayer de rendre F satisfiable.

    Pour ce faire, on vérifie si la valuation actuelle satisfait F : si oui, on sait que F est satisfiable, si non, on sait qu'il y a au moins une clause non satisfaite. On en trouve une avec trouve_clause_fausse et on essaie de la rendre vraie : ce n'est possible que si elle contient un littéral positif puisque ses littéraux négatifs ont tous été rendus vrais par la valuation initiale (et donc si l'un d'eux est faux, c'est suite à une nécessité imposée par le traitement d'un littéral positif précédent).

```
let hornsat (f:hornformule) :valuation option =
  let nb_variables = fst f and clauses = snd f in
  let pas_sat = ref false and sat = ref false and v = Array.make nb_variables false in
  while not !pas_sat && not !sat do
    match trouve_clause_fausse clauses v with
    |None -> sat := true
    |Some c -> begin match c.csqc with
               |None -> pas_sat := true
               |Some i -> v.(i) <- true
               end
  done;
  if !pas_sat then None else Some v
```

11. La fonction evalue est linéaire en la taille de la clause qu'elle prend en entrée. La fonction trouve_clause_fausse applique evalue à chacune des clauses de la liste L en entrée au pire cas, d'où une complexité en $O(m \times l)$ où m est le nombre de clauses de L et l la taille maximale d'une clause.

    An pire cas, on est amené à appeler trouve_clause_fausse autant de fois qu'il y a de variables distinctes dans la formule en entrée de hornsat. D'où une complexité pour hornsat sur une formule F en $O(nml)$ où n est le nombre de variables dans F, m son nombre de clauses et l la taille maximale d'une de ses clauses.

    On peut aussi être plus grossier et majorer cette complexité par un $O(N^3)$ où N est la taille de la formule en entrée puisque m, n et l y sont toutes inférieures. Dans tous les cas, on doit retrouver une complexité polynomiale en la taille de l'entrée, comme en 5.